Communicating with ALOHA

December 2007

This document is for programmers and describes how to program your application to communicate with ALOHA. It includes information about communicating with the Windows version of ALOHA using the NOAA_32.DLL or the Macintosh version of ALOHA using Apple Events.

There are two common types of applications that wish to communicate with ALOHA:

1. **Mapping Applications** can draw ALOHA's threat zone and communicate threat point information. Section 1 of this document describes how to read ALOHA's threat zone "status" file, how to communicate a user specified threat point location to ALOHA, and how to ask ALOHA to notify your application about new threat zones and threat locations.
2. **Meteorological Applications** can communicate meteorological data directly to ALOHA rather than have ALOHA take over the serial port. Section 2 of this document describes the messages and formats for sending atmospheric information to ALOHA.

Microsoft Windows Version: Communication via NOAA_32.DLL

The process of inter-application communication on the IBM is accomplished through the NOAA-written 32 bit DLL called NOAA_32.DLL. (Note: Versions of ALOHA prior to version 5.3 used a 16-bit DLL called NOAA_16.DLL). To communicate with other applications, you will need to use the following basic calls (see Appendixes 3 and 4 for more details):

1. **Register** with the DLL by calling NERegister().
2. **Send messages** by calling NESendMessage() with the message string you wish to send. See Appendix 2 for the details of message strings.
3. **Receive messages** by calling NEGetNextMessage(). It is recommended that you call this function on idle. An alternative method is described in the note following NERegister() in Appendix 3.
4. When quitting your application, it is polite to say 'BYE ' to those applications you have sent messages to, and then unregister with the DLL by calling NEBye().

Macintosh Version: Communication via Apple Events

The process of inter-application communication on the Macintosh is accomplished through Apple Events. All messages are sent with event class 'NOAA' and apple event ID 'AEVT'. We call these special apple events "NOAA messages". Parameters for these NOAA messages are just Apple Event parameters and so are characterized by their 4 char OSType keyword. The data for each keyword parameter is passed as typeChar data. Which keyword parameters are present varies depending on the message, but the '**MSSG**' keyword parameter is required and can be thought of as the message type. We use the phrase an "HOLA message" for a class 'NOAA' apple event which has the value of "HOLA" in parameter specified by the keyword 'MSSG'.

Your program will need to install an apple event handler to deal with 'NOAA' messages using something like

```
AEInstallEventHandler(OSTYPE_NOAA,'AEVT',
        NewAEEventHandlerProc(HandleNOAAEvent), 0, false);
```

and must retrieve parameters using a function using code like

```
AEGetParamPtr(ae, keyWord, type, actualType, ptr, maxSize, actualSize);
```

where type = typeChar and keyWord is a 4 char code such as 'MSSG'.

Section 1: ALOHA's Threat Zone and Threat Point Status

Note: For more information about these status files, see the document called "ALOHA Pass Files".

While running, ALOHA maintains a file giving the status of the threat zones being displayed to the user. ALOHA deletes this file as soon as the information (i.e., threat zone) no longer corresponds to what is being displayed to the user. Your application should not display a threat zone whenever the file does not exist. The threat zone file is described in Appendix 1.

The coordinates of ALOHA's Threat Point can be requested from ALOHA using inter-application communication as described later in this document. Your application can also ask ALOHA to use your values for the threat point coordinates using inter-application communication.

Note: The Windows version of ALOHA maintains a threat point information file that contains only one line with two numbers in it, representing <Meters East> < Meters North> from the source point. It is called "ALO_CLP.PAS."

Requesting Notification from ALOHA

In order to request notification messages from ALOHA, you must do the following:

1. Register your application with NOAA_32.DLL (if using MS Windows version),
2. Send ALOHA a notification request (NFTY message), and
3. Remember to say 'BYE ' to ALOHA when quitting.

Sending ALOHA a Notification Request

To ask ALOHA to notify your application when threat zone or threat point coordinates change, send a 'REGA' (register application) message to ALOHA and include a 'NTFY' parameter with any value starting with "y" (such as "YES"). As a short cut, you can simply add the 'NTFY' parameter to an 'HOLA' or 'OKHI'. See Appendix 2 for a specific example of the register message.

Terminating and Restarting Notifications

To unregister, send a 'REGA' messages with 'NO' as the 'NTFY' parameter. To register, just resend a 'REGA' messages with 'YES' as the 'NTFY' parameter.

Messages You Will Receive from ALOHA

'**BYE** ' -- when ALOHA quits (Note: There is a space character in this 4 char message.)

'**NTF!**' -- when the threat zone has changed, ALOHA will send you a 'NTF!' message with these parameters:
'**YRPS**' -- your PSIG
'**EDIS**' -- in theory, either "Y" or "N" indicating if the dispersion menu is enabled (i.e., if ALOHA has a source set and the user can choose the threat zone menu).
If there is a threat zone, the '**FILE**' parameter gives the path name to the ALOHA threat zone file.

If a valid threat point has been specified, the following parameters will be included:
'**MTRN**' -- the meters north for the threat point
'**MTRE**' -- the meters east for the threat point

Other Optional Messages to ALOHA

At anytime, you can send 'NTF?' to request the notification information from ALOHA. Note: You need not have registered with ALOHA in order to do this.

Section 2: Meteorological Information

Meteorological venders who find ALOHA's serial port implementation limiting can write applications that communicate SAM information directly to ALOHA.

In order to use your application as the meteorological information provider for ALOHA, you must do the following:

1. Register your application with NOAA_32.DLL (if using MS Windows version).
2. Register your application with ALOHA by sending a 'REGA' message.
3. Inspect messages sent by ALOHA to know if ALOHA is accepting SAM data.
4. Send SAM information to ALOHA using **'SAM!**' messages when ALOHA is accepting SAM data.
5. Remember to say '**BYE** ' to ALOHA when quitting.

Registering Your Application with ALOHA

Send ALOHA a '**REGA**' (register application) message and include a '**REGA**' parameter with value of '**SAMA**' (SAM application) indicating your application can communicate SAM information to ALOHA.

If ALOHA has received this message before the user selects the SAM menu item, the user is asked if they want to use your application instead of the serial port. If they chose your application, ALOHA will not read from the serial port, but will instead read the information from the '**SAM!**' messages it receives.

When registering a SAM application with ALOHA, you can include optional parameters to specify the SAM station height. The 'WRFH' parameter is a string containing the value of the height and the 'WRFU' parameter is used to specify the units for the height (one of the four values "METERS", "FEET", "INCHES", or "CENTIMETERS"). For example, 'WRFH' = "10.0" and 'WRFU' = "METERS".

Messages You Will Receive from ALOHA

'**SAM!**' **--** a message to indicate your application should start or stop transmitting data. You will receive this message with "START" in the '**DATA**' parameter after the user selects the SAM menu item to specify that that SAM station should be used. Your application will receive this message with "STOP" in the '**DATA**' parameter when the user switches back to user inputted atmospheric data.
'**BYE** ' **--** when ALOHA quits. (Note: There is a space character in this 4 char message.)

Sending Meteorological Data to ALOHA

The '**SAM!**' messages your application sends to ALOHA simply substitute for reading from the serial port. The data string sent in the '**DATA**' parameter is treated by ALOHA exactly the same as if the string was received through the serial port. In particular, ALOHA expects the same format as used for SAM transmissions to the serial port.

Appendix 1: ALOHA's Threat Zone File

The threat zone (pass) file is called "ALO_FTP.PAS" and contains drawing instructions (in meters east and north) using move to, line to, and arc commands. The threat zone and the confidence lines are delineated via keywords FOOTPRINT and CONFIDENCE LINES. The rest of the lines are distinguished by the first letter of the line.

T -- very brief summary text displayed in a program like CAMEO DOS
t -- the text ALOHA would add to the top of a printout of the threat zone
M < x > < y > -- move to x y (in meters east, meters north from the source) (pen up)
L < x > < y > -- draw line to x y (in meters east, meters north from the source)

The following is an example of an ALOHA "ALO_FTP.PAS" file with 3 threat zones and only one set of confidence lines. Note: In versions of ALOHA prior to 5.4, the term footprint was used instead of threat zone, and FOOTPRINT is still used in the source code.

```
T ALOHA heavy gas threat zone
T CHLORINE
FOOTPRINT [LOC: 0.5 ppm = AEGL-1(60 min)] [COLOR: YELLOW]
M 4.5 -0.4
L 4.5 0.0
L 4.5 0.4
 <more lines of data>
L 16.7 -14.9
L 4.5 -0.4
CONFIDENCE LINES [LOC: 0.5 ppm = AEGL-1(60 min)] [COLOR: YELLOW]
M -4.5 0.4
L -22.7 -2.5
L -36.4 -5.8
 <more lines of data>
L 21.9 -6.4
L 4.5 -0.4
t  Time: April 5, 2006 & 1423 hours CDT (using computer's clock)
t  Chemical Name: CHLORINE
t    Carcinogenic risk - see CAMEO
t  Wind: 5 knots from 5° true at 3 meters
t  THREAT ZONE:
t    Model Run: Heavy Gas
t    Red   : 1331 yards --- (20 ppm = AEGL-3(60 min))
t    Orange: 1.7 miles --- (2 ppm = AEGL-2(60 min))
t    Yellow: 2.8 miles --- (0.5 ppm = AEGL-1(60 min))
FOOTPRINT [LOC: 2 ppm = AEGL-2(60 min)] [COLOR: LIGHT ORANGE]
M 4.5 -0.4
L 4.5 0.0
<more lines of data>
L 29.2 -25.0
L 16.7 -14.9
L 4.5 -0.4
FOOTPRINT [LOC: 20 ppm = AEGL-3(60 min)] [COLOR: RED]
M 4.5 -0.4
L 4.5 0.0
L 4.5 0.4
<more lines of data>
L 24.8 -24.6
L 16.7 -14.9
L 4.5 -0.4
```

See the document called "ALOHA Pass Files" for more information.

Appendix 2: Message String Format

Messages are based on 4 character "messages" and 4 character "parameter keys". The parameters or keys are separated by a vertical tab (ascii char 11) which is denoted as <vt> below. Message keys are all capitals (and case sensitive).

Your application needs to identify itself with a 4 character "signature" and a 4 character "psuedoSignature".

Note: On the Macintosh, applications have a hidden file attribute, called the signature, that the system uses to match files with their creator. The psuedoSignature allows a single executable to have several different "identities" on the Macintosh whereas the signature is fixed and predetermined for such applications as HyperCard and FoxPro.

On the IBM, you are free to choose any signature and psuedoSignature you like.

ALOHA's signature is ALH5.

On the IBM, messages are one long c-string of keys and values.

The first few keys are required and must be in this order:

MSSG = the message key (in the example below, let's use "REGA" register application)

SIGN = your signature (in the example below, let's use "FRED")

PSIG = your psuedoSignature (in the example below, let's use "PETE")

XTRA = an extra string (you can usually pass "")

These required values are followed by the optional parameter keys and values. Different messages require different parameters. The order of the optional parameters is not specified and it is recommended you put the short parameters toward the front of the message string to speed parsing.

Example messages to ALOHA (which has a signature of "ALH5").

Call NESendMessage("ALH5",messageStringBelow,FALSE,NULL,NULL));
with the messageStringBelow.

1. To register you application with ALOHA, send this message to 'ALH5':

   ```
   MSSG<vt>REGA<vt>SIGN<vt>FRED<vt>PSIG<vt>PETE<vt>XTRA<vt><vt>NTFY<vt>YES
   ```

2. To get ALOHA to use your values for the Threat point, send ALOHA ('ALH5') a 'CDP!' message with the meters east 'MTRE' value and meters north 'MTRN' value.

   ```
   MSSG<vt>CDP!<vt>SIGN<vt>FRED<vt>PSIG<vt>PETE<vt>XTRA<vt><vt>MTRE<vt>35.2
   <vt>MTRN<vt>56.89
   ```

An example of a notification message ('NTF!') you will receive from ALOHA is:

```
MSSG<vt>NTF!<vt>SIGN<vt>ALH5<vt>PSIG<vt>ALHA<vt>XTRA<vt><vt>YRPS<vt>PETE
<vt>EDIS<vt>Y<vt>MTRE<vt>1000<vt>MTRN<vt>-23.6<vt>FILE<vt>C:\ALOHA\ALO_FTP.PAS
```

Appendix 3: NOAA_32.DLL

Note: NOAA_32>DLL uses C-calling conventions rather than the more standard WINAPI calling conventions.

```
long NERegister(
char* sig,  // 4 char string to identify your application, e.g. ALOHA is "ALH5"
HWND mainHWnd,
char* className, // the class name of your application (if known)
char* messageStringForHola, // unimplemented, pass NULL
char* humanName, //to specify your application, e.g.ALOHA uses "ALOHA"
char* wakeUpTopicString,
char* fullPath, // of your application

UINT wakeUpMessage,// the message  parameter to be used in SendMessage()
WORD wakeUpWord, // the WORD parameter to be used in SendMessage()
LONG wakeUpLong // the LONG parameter to be used in SendMessage()
);
```

Important Note: If you specify a non-zero wakeUpMessage parameter, then when the DLL receives a message for your application, it will send your main window handle a message using the Window's SendMessage() function. Pass zero for wakeUpMessage if you do not want a message sent to your main window handle. We recommend avoiding the use of a wakeup message when you are able to check for your messages on idle. Checking for messages on idle is a very safe way to handle the whole situation. If you do request a wakeup message, be certain to code for the possibility that the DLL will send you that wakeup message again (for another message) while your code is handling the first message. We had trouble with this in Foxpro. Finally, as a special case to support CAMEO Windows (which was written in Foxpro 2.6), if the wakeUpMessage is WM_COMMAND, then the parameters have special meaning and the DLL will send a message corresponding to selecting a menu item. For details, see Appendix 4.

```
long NEBye(char* sig,
HWND mainWHnd,
char* className,
char*  messageStringForBye  // unimplemented, pass NULL);

long  FAR PASCAL _export NESendMessage(
char* toSig, // the 4 character code of the application you want to talk to
char* messageString,
UINT launch, // unimplemented, pass FALSE
char* humanName,  // unimplemented, pass NULL
char* possibleFullPath  // unimplemented, pass NULL);

BOOL NEAppIsRunning(char* sig);

long NEGetNextMessageLength(char* sig);

BOOL NEGetNextMessage(
char*sig, char* s,
long maxLength);
// returns TRUE if it successfully filled in your parameter

long NEPeekMessage(
int n, char* sig, char* s,
long maxLength,long offsetIntoMessage);
// returns the length of the next message, so you can allocate enough space.  Your
application can probably just allocate 256 bytes and forget about it.
```

Appendix 4: Our Application Wakeup Implementation

Note: This section is not needed by applications that can poll the NOAA_32.dll for messages.

Motivation

Because FoxPro 2.6 (a 16-bit application) did not support an "on idle" hook, we needed a way to get FoxPro's attention when the NOAA_16.DLL had a message for FoxPro. Our solution was to send the main window handle a message simulating the selection of a menu item. The selection of a menu item is a supported way to get FoxPro to execute our script to retrieve and handle the message. The only problem we had to overcome was the fact that we did not know the ID of the menu item because the FoxPro application creates the menu and assigns the ID.

To give the exact example used in CAMEO, we installed our wakeup menu under a "Sharing" menu which had a hierarchical submenu for "MARPLOT" which had the wakeup menu item as an item. The Sharing menu looks something like:

```
Sharing
   ALOHA
   MARPLOT
      Get Info
      Link
      -(the wakeup menu that executes the code in CAMEO)
      Go to MARPLOT
   SitePlan
```

We had control over all of the items under the Sharing menu, so we could rely on the position of the wakeup menu item under the Sharing menu. We solved the problem of finding the main menu "group" by passing the text of the main menu (e.g., "Sharing") as a parameter. To find the submenu item, we pass numbers for the position.

Implementation

To request that NOAA_32.DLL send your application simulating a menu item being selected when you have a message, you need to call NERegister() with the following parameters.

`wakeUpTopicString` -- the text of the menu your menu item is under. In the case of CAMEO this was "Sharing".
`wakeupMessage` -- WM_COMMAND (= 256+16+1).
`wakeupWord` -- itemNum of the first submenu. (In the example shown above, this would be the 0-relative number of the MARPLOT menu, which is 1 (**).
`wakeupLong` -- subitemNum of the wakeup menu item. In the example shown above, this would be 2 (= 3-1) (**).

(**) Note: The itemNum and subitemNum are 0-relative positions (i.e., the counting starts with 0, so the first position is number 0, the second position is number 1, etc.)

Note: If your wakeup menu item is not under a hierarchical submenu, but is directly under the main menu "group", pass 0 for the subitemNum.

For specifics on what the DLL code actually does, you can examine the function on the next page that is used by the DLL to find the menu number.

(code from NOAA_32.DLL)

```
WORD GetMagicMenuID(HWND hWnd, char* topicName, int itemNum, int subitemNum)
{
   char str[64];
   WORD id = 0;
   int i, len;
   HMENU topicMenu, itemMenu, mainMenu = GetMenu(hWnd);

   for (i = 1, topicMenu = (HMENU)1 ; i < 15 && mainMenu && topicMenu ; i++) {
      // look for the sharing menu
      len = GetMenuString(mainMenu, i, str, 63, MF_BYPOSITION);
      if (StrMatches(str, topicName)) {
         topicMenu = GetSubMenu(mainMenu, i);
         itemMenu = GetSubMenu(topicMenu, itemNum);
         if (subitemNum == 0)
            id = GetMenuItemID(topicMenu, itemNum);
         else
            id = GetMenuItemID(itemMenu, subitemNum);
         break;
      }
   }

   return id;
}
```

Appendix 5: IBM Code from ALOHA

```
#ifdef IBM ///{

HINSTANCE gNoaaDllInst = 0;

void LoadNoaaDll(void)
{
   if(gNoaaDllInst)
      return; // already loaded

   {
      Boolean doCopy = FALSE;
      char theirPath[256], myPath[256];
      MODIFICATIONDATE theirTime, myTime;

      GetWindowsDirectory(theirPath, 255);
      strcat(theirPath, "\\NOAA_32.DLL");

      OurDirectoryWithDelimiter(myPath);
      strcat(myPath, "NOAA_32.DLL");

      theirTime = GetModificationDate(theirPath);
      myTime = GetModificationDate(myPath);


      if (theirTime == 0)
         doCopy = TRUE; // their file does not exist
      else if (myTime == 0)
         doCopy = FALSE; // my file does not exist
      else if (theirTime < myTime)
         doCopy = TRUE; // my file is more current than their file

      // for now, we'll just try to copy it and not worry about whether it fails
      // it may be that if it fails because the DLL is in use, we should tell the
      // user to quit other applications so the DLL can be updated

      if (doCopy)
         CopyFile(myPath, theirPath, FALSE);

      gNoaaDllInst = LoadLibrary(theirPath);


   }
}
```

```
void CallNERegister(void)
{
   char sigStr[6];
   char fullPath[256];
   char humanName[64];
   FARPROC proc=NULL;
   LoadNoaaDll();
   if((UINT)gNoaaDllInst > 32)
   {
      //we have the library
      proc = GetProcAddress(gNoaaDllInst,"NERegister");
      if(proc)
      {
         OSType2String(gMySignature,sigStr);
         GetPathToAlohaPlusExtension(fullPath);
         my_getindstring(humanName,1000,1); //ALOHA
         (*proc)(  (char*) sigStr,
                   (HWND) global_stdglobal_ptr->main_wnd_hdl,
                   (char*) MAIN_CLASS_NAME,
                   (char*) NULL, //messageStringForHola,
                   (char*) humanName,
                   (char*) "ALOHA.exe",
                   (char*) fullPath,
                   (UINT) SA_APPTASK,
                   (WORD) 0,
                   (LONG)0);
      }
   }

}

void CallNEBye(void)
{
   char sigStr[6];
   FARPROC proc=NULL;
   if((UINT)gNoaaDllInst > 32)
   {
      //we have the library
      proc = GetProcAddress(gNoaaDllInst,"NEBye");
      if(proc)
      {
         OSType2String(gMySignature,sigStr);
         (*proc)(  (char*) sigStr,
                   (HWND) global_stdglobal_ptr->main_wnd_hdl,
                   (char*) MAIN_CLASS_NAME,
                   (char*) NULL); //messageStringForBye,
      }
      FreeLibrary(gNoaaDllInst);
      gNoaaDllInst = NULL;
   }
}
```

```
OSErr CallNESendMessage(char* toSigStr, char* messageStr)
{
   //OSErr  MY_CALLBACK NESendMessage(char* toSig,char* messageString,Boolean launch,char* 
humanName, char* possibleFullPath)

   FARPROC proc=NULL;
   OSErr err = -1;
   if((UINT)gNoaaDllInst > 32)
   {
      //we have the library
      proc = GetProcAddress(gNoaaDllInst,"NESendMessage");
      if(proc)
      {
         err = 
(OSErr)(*proc)((char*)toSigStr,(char*)messageStr,(Boolean)FALSE,(char*)NULL,(char*)NULL);
      }
   }
   return err;
 }

Boolean CallNEAppIsRunning(char* toSigStr)
{

   FARPROC proc=NULL;
   Boolean isRunning = FALSE;
   LoadNoaaDll();
   if((UINT)gNoaaDllInst > 32)
   {
      //we have the library
      proc = GetProcAddress(gNoaaDllInst,"NEAppIsRunning");
      if(proc)
      {
         isRunning = (Boolean)(*proc)((char*)toSigStr);
      }
   }
   return isRunning;
 }
```

```
OSErr HandleNEMessage(void)
{
   // check for a message and handle it
   char sigStr[6];
   FARPROC proc=NULL;
   OSErr err = -1;
   long len;
   LoadNoaaDll();
   if((UINT)gNoaaDllInst > 32)
   {
      //we have the library
      OSType2String(gMySignature,sigStr);
      proc = GetProcAddress(gNoaaDllInst,"NEGetNextMessageLength");
      if(proc)
      {
         len = (long)(*proc)((char*)sigStr);
         if(len > 0)
         {
            // we have a message
            proc = GetProcAddress(gNoaaDllInst,"NEGetNextMessage");
            if(proc)
            {
               long maxLength = len+1;
               char* messageString = NewPtrClear(maxLength);
               if(messageString== nil) MessageBeep(5);// memory error
               else
               {
                  Boolean gotIt;
               gotIt =
(Boolean)(*proc)((char*)sigStr,(char*)messageString,(long)maxLength);
               if(gotIt) HandleNOAAEvent(&messageString,nil,0);
               else MessageBeep(5);
               DisposePtr(messageString);
               }
            }
         }
      }
   }
   return err;
 }
```

Appendix 6: Other Messages to/from ALOHA

Messages to ALOHA

(all messages include 'SIGN', 'PSIG', 'MSSG' and 'XTRA' parameters)

Message	Parameters	Description
'BYE '		The friend application is quitting. ALOHA will not send any more messages to it.
'HOLA'		Initial greeting from a friend application. The friend sends this message to ALOHA when it starts up and sees that ALOHA is running. This tells ALOHA that the friend is alive and ready to handle messages. ALOHA responds with an OKHI message. If the sender is MARPLOT, ALOHA also sends a MENU message to install ALOHA's sharing menu.
	'VERS'	"2" or greater means you are using the updated IAC messages.
	'NAME'	Name of the friend application.
	'PATH'	Full path to friend application's executable file.
	'DOC '	(optional) Full path to default document to be opened when ALOHA launches the friend.
'OKHI'		Acknowledge receipt of HOLA from ALOHA. The friend application has received an HOLA message from ALOHA, and is acknowledging so that ALOHA will know the friend is running. If the sender is MARPLOT, ALOHA also sends a MENU message to install ALOHA's sharing menu.
	'VERS'	"2" or greater means you are using the updated IAC messages.
	'NAME'	Name of the friend application.
	'PATH'	Full path to the friend application's executable file.
	'DOC '	(optional) Full path to default document to be opened when ALOHA launches the friend.
'MENU'		Install sub-menu in ALOHA's Sharing menu. This adds a new sub-menu to ALOHA's Sharing menu. If a sub-menu with the same name already exists, it is replaced with the new menu. (Thus, a friend application can send a MENU message each time it greets ALOHA, without worrying about duplicating the menu.) Menus installed in ALOHA are automatically saved by ALOHA. They can be used later, even when the friend application is not running. In this case, ALOHA launches the friend application before sending it the MHIT message.
	'VERS'	"2" or greater; needed to show ALOHA you are using the updated IAC messages.
	'NAME'	Name of menu.
	'ITMS'	Return-delimited string of menu items.
	'PATH'	Full path to the friend application's executable file.
	'DOC '	(optional) Full path to default document to be opened when the friend is launched.
'FRWD'		Your application can ask ALOHA to bring it to the foreground using this message. There are a number of technical issues involved in getting an application to come automatically to the foreground. These issues are different for each platform/system. In some cases, when an application wants to bring some application (often itself) to the foreground, it is easier (and sometimes more polite) to ask another application to do the job.
	'WHO '	Signature of application to bring forward (usually the sending app itself).
	'NAME'	Name of application to bring forward; on Windows, NAME should be the title of your main window, or the name of your main window class.
		NOTE: On the Macintosh, it is sometimes better to use the Notification Manager and let the user bring you forward.
'CHM?'		A request for ALOHA to send a CHM! message indicating the current ALOHA chemical.
'CHEM'		A request for ALOHA to select a chemical.
	'NOAA'	(optional) A string containing the NOAA number of the chemical.
	'CAS '	(optional) A string containing the CAS number of the chemical (as a number with no dashes).
	'NAME'	(optional) A string containing the name of the chemical.
'CTY?'		A request for ALOHA to send a CTY! message indicating the current ALOHA city.
'INFO'		A request for ALOHA to come forward and show information on the threat zone or the threat point.
	'LIST'	If this is "2", the concentration window is presented, otherwise the text summary window is presented.
'NTF?'		A request for ALOHA to send a NTF! message indicating the current threat zone information.

'REGA'		A request for ALOHA to register an application as a "Mapping" or "SAM" application (based) on the REGA parameter.
	'REGA'	Specifies the type of application. One of the two values MAPA (for Mapping application) or SAMA (for SAM application)
	'WRFH'	Optional parameter (wind reference height) for SAM application. A string giving the value of the height of the station (e.g., "3.0"). Note: Units are specified by the 'WRFU' parameter.
	'WRFU'	Optional parameter (wind reference height units) for SAM application. A string specifying the units for the 'WRFH' parameter for SAMA case, one of the four values: METERS, FEET, INCHES, CENTIMETERS.
'SAM!'		A request for ALOHA to use the given atmospheric data.
	'DATA'	A string using exactly the same format as would be transmitted by a SAM station through the serial port.
'VER!'		A response to ALOHA's 'VER?' message indicating the current version of sending application.
	'VNUM'	A string with the version number (e.g. "3.3.3" for MARPLOT 3.3.3)

Other Messages Sent to ALOHA by MARPLOT:

MHIT	ALOHA assumes it is its menu in MARPLOT, but does not check the sender.
CPT!	MARPLOT click point response to ALOHA's 'CPT?' message. ALOHA assumes that the sender was MARPLOT, but that is not checked.
IMP!	A response to ALOHA's ‘IMP2' import message. ALOHA just records the IDS.
OVL!	A response to ALOHA's ‘MKOV' make overlay (layer) message. ALOHA simply records the ID.
OBID	A response to ALOHA's ‘IMPT' import message. ALOHA just records the IDS.
INFO	ALOHA checks that the sender is MARPLOT, then checks to see if there is one object and that its data matches threat point. If so, shows the Threat at Point window, else it shows the Text Summary.
CLOS	If from MARPLOT, ALOHA clears the global ID's.
ULK?	ALOHA sends MARPLOT an alert saying you cannot unlink ALOHA objects. (This is no longer used)

Messages Sent by ALOHA

(all messages include 'SIGN' ('ALH5'), 'PSIG' ('ALHA'), 'MSSG' and 'XTRA' parameters)

Message	Parameters	Description
`'BYE '`		ALOHA is quitting. This is to inform you that if you plan to send ALOHA any more messages, you will have to wait until it gets started again (perhaps by your launching it) and sends you an HOLA message. This message is sent to all friend applications that are currently running.
`'HOLA'`		Initial greeting from ALOHA. ALOHA sends this message to all running friend applications when it starts up. This tells friend applications that ALOHA is alive and ready to handle messages. ALOHA's friend applications are MARPLOT, CAMEO, and any application that has ever said HOLA to ALOHA. The list of friends is saved in the ALOHA.PRF file.
	`'NAME'`	"ALOHA"
	`'PATH'`	Full path to ALOHA.
	`'DOC '`	Empty string; provided for consistency with other applications.
	`'VERS'`	"2"
	`=====>`	When you get an HOLA message from ALOHA, you should respond with an OKHI message.
`'OKHI'`		Acknowledge receipt of HOLA. ALOHA has received an HOLA message from an application that just started up and is acknowledging so that the other application will know ALOHA is alive. You should treat an incoming OKHI message the same as an incoming HOLA message; they give the same information but you will get one or the other depending on whether your application or ALOHA is started first.
	`'NAME'`	"ALOHA"
	`'PATH'`	Full path to ALOHA.
	`'VERS'`	"2"
	`'DOC '`	Empty string; included for consistency with other applications.
`'FRWD'`		A request to bring an application forward. ALOHA will only ask you to bring either your own application forward or to bring ALOHA forward. There are a number of technical issues involved in getting an application to come automatically to the foreground. These issues are different for each platform/system. In some cases, when an application wants to bring some application (often itself) to the foreground, it is easier (and sometimes more polite) to ask another application to do the job.
	`'WHO '`	Signature of application to bring forward.
	`'NAME'`	Name of application to bring forward; on Windows, NAME should be the title of your main window, or the name of your main window class.
		NOTE: On the Macintosh, it is sometimes better to use the Notification Manager and let the user bring you forward.
`'CDP!'`		Tells ALOH A to use this as the Threat Point (previously known as the Conc/Dose Point).
	`'MTRE'`	A string containing the meters east (decimal value)
	`'MTRN'`	A string containing the meters north (decimal value)
`'CHM!'`		The answer to a request for ALOHA to indicate which chemical is selected in ALOHA.
	`'NOAA'`	A string containing the NOAA number of the chemical. (User-added chemicals have 0 for a NOAA number)
	`'CAS '`	A string containing the CAS number of the chemical (as a number with no dashes). (User-added chemicals have 0 for a CAS number)
	`'NAME'`	A string containing the name of the chemical.
`'CTY!'`		The answer to a request for ALOHA to indicate which city is selected in ALOHA.
	`'NAME'`	A string containing the name
	`'LATD'`	A string containing the degree value of the latitude. (integer value)
	`'LATM'`	A string containing the minute value of the latitude. (decimal value)
	`'LNGD'`	A string containing the degree value of the longitude. (integer)
	`'LNGM'`	A string containing the minute value of the longitude. (decimal value)
`'NTF!'`		Notification of a change in the threat zone. See section 1 above.
`'RIDS'`		Sent to CAMEO (CAMO) when the user asks to see RIDS information on the chemical currently selected in ALOHA.
	`'NOAA'`	A string containing the NOAA number of the chemical. (User-added chemicals have 0 for a NOAA number.)
	`'CAS '`	A string containing the CAS number of the chemical (as a number with no dashes). (User-added chemicals have 0 for a CAS number.)
	`'NAME'`	A string containing the name of the chemical.
`'SAM!'`		Start or stop sending SAM data to ALOHA
	`'DATA'`	A string containing ‘START' to start sending data and ‘STOP' to stop sending data to ALOHA.

Other Messages Sent to MARPLOT by ALOHA:

See the MARPLOT tech doc for more information.

`CPNT`	To get MARPLOT's current click.
`DELO`	To delete our objects.
`DLOV`	To delete the ALOHA layer (overlay).
`IMPT`	To display the threat zone etc.
`MENU`	To install a sharing menu in MARPLOT.
`MKOV`	To create the ALOHA layer (overlay) and get the layer ID.